

计算机组成

# 有限状态机

高小鹏

北京航空航天大学计算机学院

## 做中学 (learning by doing) <sup>1/6</sup>

□ 目标：构造一个2位连续计数器，计数模式如下：

◆ 00⇒01⇒10⇒11⇒00⇒01⇒10⇒11⇒....

□ 分析1：必须有某种**时间**因素

◆ 但是：多快多慢，不清楚（也不需要清楚）

结论1：  
需要有一个Clk

□ 分析2：**计数器**意味着需要寄存器

◆ 因为必须能【存储】信息

◆ 因为必须在【下一个】【特定时刻】未来临前【不应改变】

结论2：  
需要有2个寄存器

## 做中学 (learning by doing) <sup>2/6</sup>

□ 目标：构造一个2位连续计数器，计数模式如下：

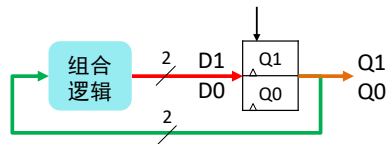
♦ 00⇒01⇒10⇒11⇒00⇒01⇒10⇒11⇒....

□ 分析3：寄存器输入值 =  $F$ (寄存器输出值)

- ♦ 1) 寄存器特性：↑前输入准备好；↑时保存输入；↑后输出
- ♦ 2) 从变化规律可以看出，寄存器当前输出值**决定**了下一个↑后的值

结论3：

必须有一个组合逻辑，用于计算，即根据 $Q_1Q_0$ 值计算 $D_1$ 和 $D_0$ 值



## 做中学 (learning by doing) <sup>3/6</sup>

□ 目标：构造一个2位连续计数器，计数模式如下：

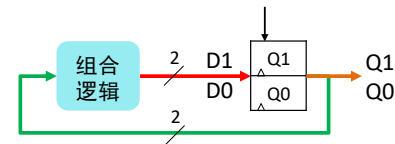
♦ 00⇒01⇒10⇒11⇒00⇒01⇒10⇒11⇒....

□ 分析4：根据变化规律，可以建立组合逻辑的**真值表**

Q1	Q0	D1	D0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

提示

D1/D0：组合逻辑的输出  
Q1/Q0：组合逻辑的输入



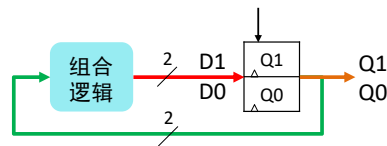
## 做中学 (learning by doing) <sup>4/6</sup>

□ 目标：构造一个2位连续计数器，计数模式如下：

◆ 00 ⇒ 01 ⇒ 10 ⇒ 11 ⇒ 00 ⇒ 01 ⇒ 10 ⇒ 11 ⇒ ....

□ 设计：

- ◆ 1) 部署2个寄存器
- ◆ 2) 根据真值表构造表达式
- ◆ 3) 根据表达式构造门电路
- ◆ 4) 连接信号线



Q1	Q0	D1	D0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

## 做中学 (learning by doing) <sup>5/6</sup>

□ 目标：构造一个2位连续计数器，计数模式如下：

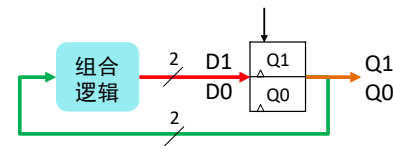
◆ 00 ⇒ 01 ⇒ 10 ⇒ 11 ⇒ 00 ⇒ 01 ⇒ 10 ⇒ 11 ⇒ ....

□ 设计：

- ◆ 1) 部署2个寄存器
- ◆ 2) 根据真值表构造表达式
- ◆ 3) 根据表达式构造门电路
- ◆ 4) 连接信号线

$$D1 = \overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0}$$

$$D0 = \overline{Q0}$$



Q1	Q0	D1	D0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

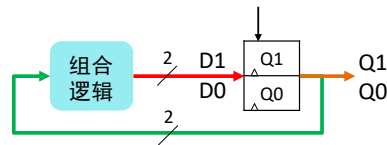
## 做中学 (learning by doing) 6/6

□ 目标：构造一个2位连续计数器，计数模式如下：

◆  $00 \Rightarrow 01 \Rightarrow 10 \Rightarrow 11 \Rightarrow 00 \Rightarrow 01 \Rightarrow 10 \Rightarrow 11 \Rightarrow \dots$

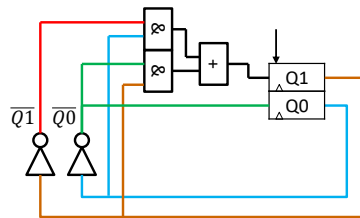
□ 设计：

- ◆ 1) 部署2个寄存器
- ◆ 2) 根据真值表构造表达式
- ◆ 3) 根据表达式构造门电路
- ◆ 4) 连接信号线



$$D1 = \overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0}$$

$$D0 = \overline{Q0}$$

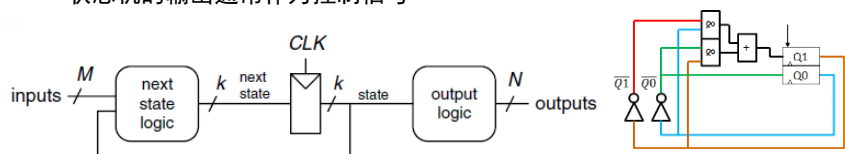


## 有限状态机概述

- 有限状态机 (Finite State Machine, FSM)：表示有限个状态以及这些状态之间的转移和动作等行为的离散数学模型
- 用途：常用于设计数字系统的控制模块
- 特点：结构简单、逻辑清晰、可靠性高

## 有限状态机概述

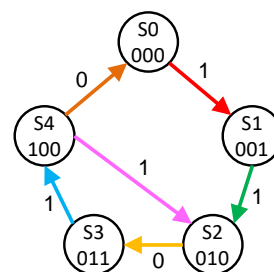
- 状态：电路所处的特定工作阶段
  - ◆ 每个状态对应一个**编码值**
- 状态寄存器：若干个触发器，其0/1输出的编码组合值就是状态
- 状态寄存器的值：取决于**次态逻辑**的输出（即寄存器的输入）
- **次态逻辑**：根据输入和当前状态编码值，计算下一个状态编码值
  - ◆ 次态逻辑也是组合逻辑（命名上与输出逻辑区分开而已）
- 输出逻辑：根据状态及输入，计算该状态下的输出值
  - ◆ 状态机的输出通常作为控制信号



## 有限状态机表示方法

- 状态机有2种表示方法
  - ◆ 状态图（State Diagram）、状态表（State Table）
  - ◆ 二者可以相互转换

状态图



状态表(用状态)

输入	S	S <sup>n</sup>
1	S0	S1
1	S1	S2
0	S2	S3
1	S3	S4
0	S4	S0
1	S4	S2

输入 现态 次态

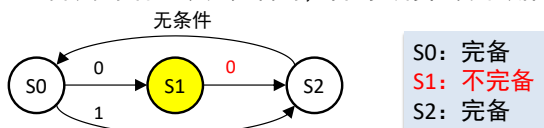
状态表(用二进制编码)

输入	Q <sub>2</sub> Q <sub>1</sub> Q <sub>0</sub>	Q <sub>2</sub> <sup>n</sup> Q <sub>1</sub> <sup>n</sup> Q <sub>0</sub> <sup>n</sup>
1	0 0 0	0 0 1
1	0 0 1	0 1 0
0	0 1 0	0 1 1
1	0 1 1	1 0 0
0	1 0 0	0 0 0
1	1 0 0	0 0 0

输入 现态 次态

## 状态机的基本开发步骤

- 1、规划状态总数
  - ◆ 不能存在有疏漏的状态
- 2、构造状态图（本质是刻画状态转化）
  - ◆ 每个状态的转移条件必须是完备的，否则会在设计缺陷！



- 3、根据{输入信号、当前状态编码、下个状态编码}构造每个寄存器的D输入信号的门电路
  - ◆ 方法：真值表→乘积项表达式→最简表达式→最简门电路
  - ◆ 寄存器个数：取决于编码方案
- 4、根据设计需求决定在当前状态输出应该取何值

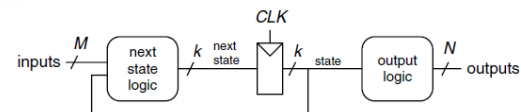
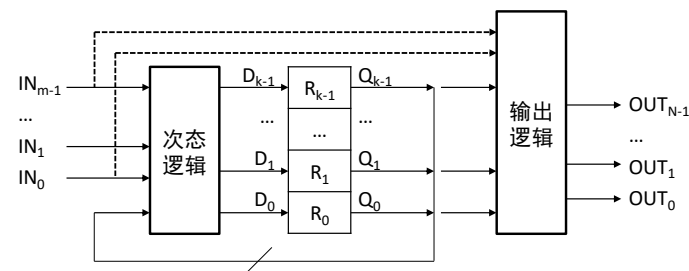


北京航  
School of Computer Science and Technology

T  
基于VerilogHDL  
的设计，主体是  
建模状态转换和  
输出逻辑。  
EDA软件自动综  
合产生门电路。

## 细化状态机内部结构

- 状态机的核心：次态逻辑计算下一个时钟周期的寄存器输入
  - ◆ 1、寄存器：由状态数量和编码方案决定
  - ◆ 2、次态逻辑：状态间的转移由次态逻辑实现



## 状态机的设计要点<sup>1/2</sup>

### □ 初始状态

- ◆ 状态机必须指定一个初始状态，以确保系统在上电后处于确定的状态

### □ 复位信号

- ◆ 复位信号的作用是将状态机强制初始化为**初始状态**

### □ 状态编码方式

- ◆ 二进制编码： $\log_2^N$ 个触发器表示N个状态
  - 节省逻辑资源，但可能产生毛刺
- ◆ 格雷编码： $\log_2^N$ 个触发器表示N个状态，但**相邻状态编码值**只有1位不同
  - 节省逻辑资源，又避免产生毛刺（状态顺序的转换中有效）
- ◆ 一位热码编码（One-Hot Encoding）：N个触发器表示这N个状态
  - 资源消耗多，但无毛刺
  - 降低次态逻辑和输出逻辑复杂度，有利于提高时钟频率

## 状态机的设计要点<sup>2/2</sup>

- 一位热码编码：可以有效提高电路的速度和可靠性

状态	二进制编码	格雷编码	一位热码编码
state0	000	<del>000</del>	0000000 <b>1</b>
state1	001	<del>001</del>	000000 <b>10</b>
state2	010	<del>011</del>	00000 <b>100</b>
state3	011	<del>010</del>	0000 <b>1000</b>
state4	100	<del>110</del>	000 <b>10000</b>
state5	101	<del>111</del>	00 <b>100000</b>
state6	110	<del>101</del>	0 <b>1000000</b>
state7	111	<del>100</del>	<b>10000000</b>

## 状态编码的VerilogHDL定义

- 状态编码的定义有两种方式: **parameter**和**'define**语句
- 示例: 假设state0, state1, state2, state3状态为00, 01, 11, 10

方式1: 用parameter参数定义  
用n个parameter常量表示n个状态

```
parameter state0=2'b00,  
           state1=2'b01,  
           state2=2'b11,  
           state3=2'b10;
```

.....

```
case (state)  
    state0:.....;  
    state1:.....;
```

.....

方式2: 用'define语句定义  
用n个宏名表示n个状态

```
'define state0 2'b00  
'define state1 2'b01  
'define state2 2'b11  
'define state3 2'b10
```

```
case (state)  
    'state0:.....;  
    'state1:.....;
```

.....

Q  
哪种方式更好呢?

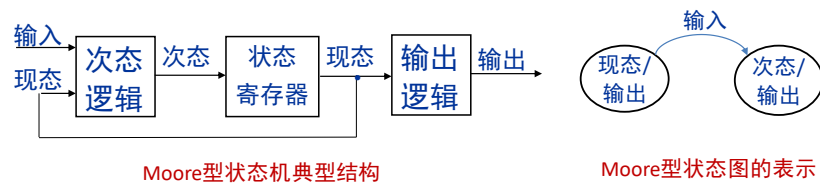
## 状态转换的描述

- 一般用**case**或**casex**语句!
- 在case语句的最后, 要加上default分支语句
  - ◆ 强制未定义状态进入某个特定状态, 例如初始状态
  - ◆ 避免状态机因为寄存器偶发错误进入未定义状态空间后无法回到正常状态空间

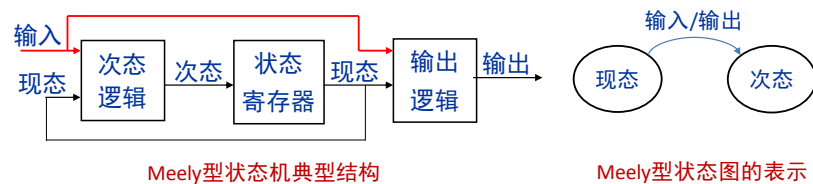


## Moore型状态机 vs Meely型状态机

- Moore型特点：输出信号仅为状态的函数，与输入信号无关

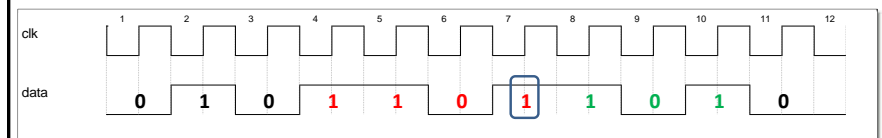


- Meely型特点：输出信号是状态与输入信号的函数



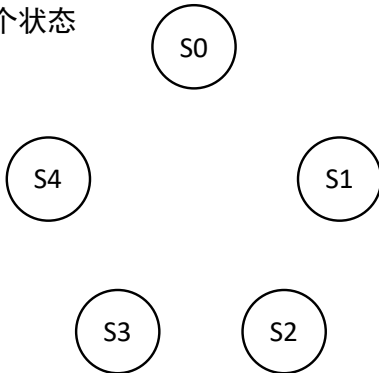
## 示例：序列检测器

- 序列检测器的1位串行输入与时钟严格同步。当检测器连续收到串行码{1101}后，检测标志输出1且持续时间为1个cycle，否则输出0。
- 分析要点
  - 1) 严格同步：意味着每个时钟周期，输入1位输入
  - 2) 设计分歧：如何理解1101101？
    - 认为是2个{1101}或1个{1101}均可。倾向于2个。



## STEP1: 状态规划

- 基本原则：先定状态，后考虑输出
  - 现阶段不考虑MOORE或MEELY！
- 思路：由于每个时钟输入1位，故可用状态对应已匹配成功的位流
- 4个cycle才能输入4位，所以需要4个状态
  - S1：匹配成功{1}
  - S2：匹配成功{11}
  - S3：匹配成功{110}
  - S4：匹配成功{1101}
- S0：无匹配成功



```
graph TD; S0((S0)); S1((S1)); S2((S2)); S3((S3)); S4((S4));
```

## STEP2: 确定状态转换

- 一般需要一个初始状态作为起始状态
  - S0: 在本例中是**最好的**初始状态
- 先确定状态迁移的**核心路径** (红色)
  - 在状态迁移箭头上标记转移条件
  - S4: 体现了前述的设计分歧
    - 当前设计能识别{1101101}中的第2个{1101}

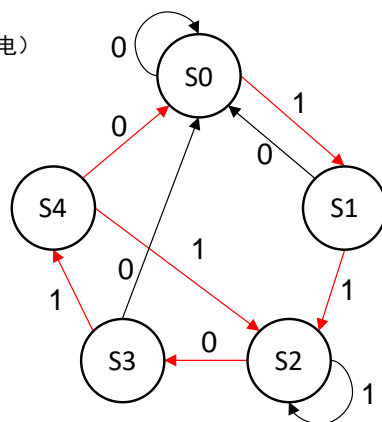
```
graph TD; S0((S0)) -- 1 --> S1((S1)); S1 -- 1 --> S2((S2)); S2 -- 0 --> S3((S3)); S3 -- 1 --> S4((S4)); S4 -- 0 --> S0; style S0 fill:#fff,stroke:#000; style S1 fill:#fff,stroke:#000; style S2 fill:#fff,stroke:#000; style S3 fill:#fff,stroke:#000; style S4 fill:#fff,stroke:#000; linkStyle 0 stroke:#f00,stroke-width:2px; linkStyle 1 stroke:#f00,stroke-width:2px; linkStyle 2 stroke:#f00,stroke-width:2px; linkStyle 3 stroke:#f00,stroke-width:2px; linkStyle 4 stroke:#f00,stroke-width:2px;
```

迁移~转换~转变~改变

### STEP3: 处理所有其他可能转移

#### □ 避免极端状态：只进不出 or 只出不进

- ◆ 只进不出：结束状态
  - 一般用于彻底停机
- ◆ 只出不进：极其原始的状态
  - 一般用于对整个电路进行复位（刚上电）



### STEP4: 增加复位（reset）信号

#### □ 多数数字电路设计都会考虑复位功能

- ◆ 确保所有数字电路在初始化阶段能被**有效初始化**，从而全系统能严格同步

#### □ 初始化阶段：大多数为**上电**后的短暂时间

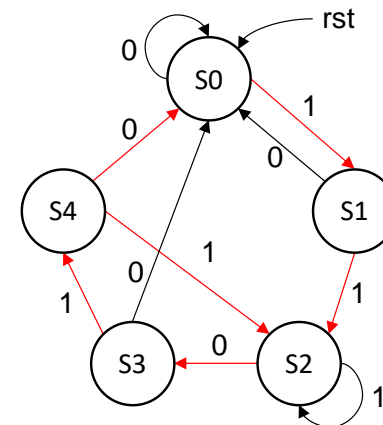
- ◆ 一般以ms为单位

#### □ Reset产生机制

- ◆ 某个系统级部件产生 or
- ◆ 某个控制器可复位其他控制器

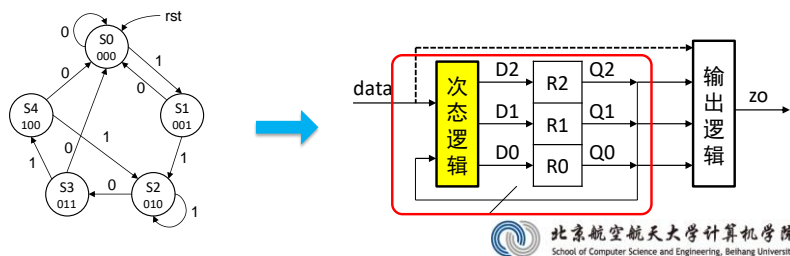
#### □ Reset电平规划

- ◆ 高电平H(1) or 低电平L(0)均可
- ◆ 芯片外部输入：一般为L有效
- ◆ 芯片内部使用：一般用H有效



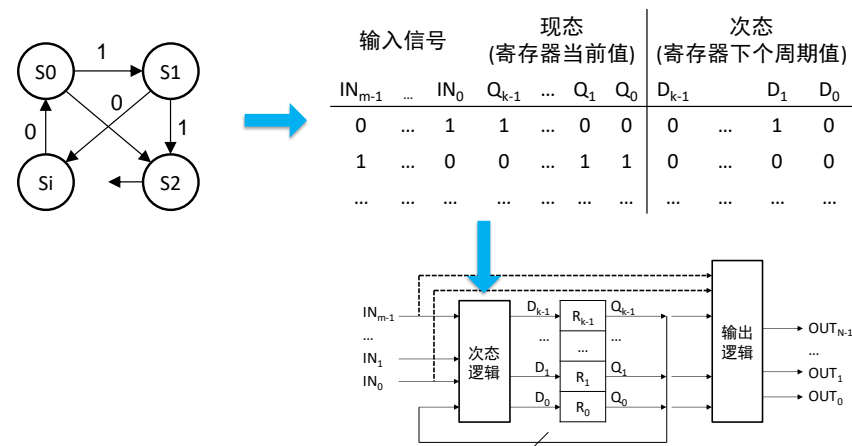
### STEP5: 构造次态逻辑的门电路

- 假设采用2进制编码方式，故需要3个寄存器，即R2、R1、R0
  - 状态编码：S0~000，S1~001，S2~010，S3~011，S4~100
- 次态逻辑就是产生寄存器输入信号的电路
  - 电路输入：寄存器的输出（即现态）、外部输入
  - 电路输出：寄存器的输入（即次态）
  - 以S1→S2为例：R[2:0]输出值为001且data为1，则次态逻辑应产生010



## 从状态图求解次态逻辑门电路的一般方法

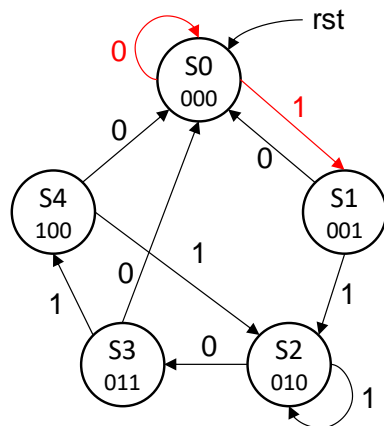
- 将状态图的每个转移都转换为真值表的一行
- 将真值表转换为表达式并化简，再根据表达式得到门电路



### STEP5: 构造次态逻辑的门电路

- 示例1: S0的自转移 (现态为000, 输入为0, 则次态为000)
- 示例2: S0→S1 (现态为000, 输入为1, 则次态为001)

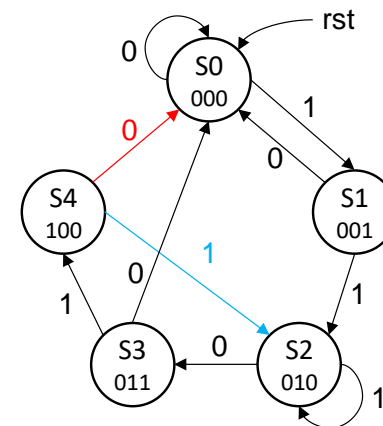
输入 data	现态			次态		
	Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	0	0
1	0	0	0	0	0	1



### STEP5: 构造次态逻辑的门电路

- 以此类推, 根据状态图建立真值表

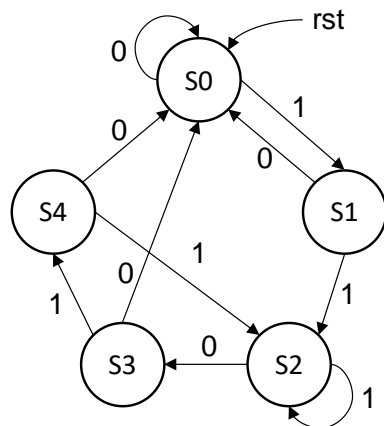
输入 data	现态			次态		
	Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	0	1	0	0	0
1	0	0	1	0	1	0
0	0	1	0	0	1	1
1	0	1	0	0	1	0
0	0	1	1	0	0	0
1	0	1	1	1	0	0
0	1	0	0	0	0	0
1	1	0	0	0	0	1
0	1	0	1	0	0	0
1	1	0	1	1	0	0
0	1	1	0	0	1	0
1	1	1	0	0	1	0



## STEP5: 构造次态逻辑的门电路

- 可靠性设计：所有的未定义状态，都强制转移至初态（S0）
  - 状态图通常不建模从非法状态转移至初始状态（非法状态可能太多！）
  - 但在状态表及后面的VerilogHDL中必须建模上述转移关系

输入	现态			次态		
data	Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	0	1	0	0	0
1	0	0	1	0	1	0
0	0	1	0	0	1	1
1	0	1	0	0	1	0
0	0	1	1	0	0	0
1	0	1	1	1	0	0
0	1	0	0	0	0	0
1	1	0	0	0	0	0
X	1	0	1	0	0	0
X	1	1	0	0	0	0
X	1	1	1	0	0	0

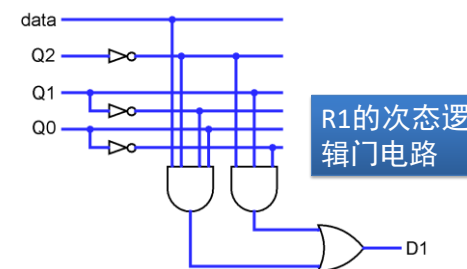


## STEP5: 构造次态逻辑的门电路

- 根据真值表得到表达式并化简
- 根据表达式给出门电路结构
  - 注意：为表达简洁，使用了非2输入与门

输入	现态			次态		
data	Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	0	1	0	0	0
1	0	0	1	0	1	0
0	0	1	0	0	1	1
1	0	1	0	0	1	0
0	0	1	1	0	0	0
1	0	1	1	1	0	0
0	1	0	0	0	0	0
1	1	0	0	0	0	0
X	1	0	1	0	0	0
X	1	1	0	0	0	0
X	1	1	1	0	0	0

$$\begin{aligned}
 D1 &= data \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + \\
 &\quad \overline{data} \cdot \overline{Q2} \cdot Q1 \cdot \overline{Q0} + \\
 &\quad data \cdot \overline{Q2} \cdot Q1 \cdot \overline{Q0} \\
 &= data \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + \\
 &\quad \overline{Q2} \cdot Q1 \cdot \overline{Q0}
 \end{aligned}$$



## STEP6: 产生输出信号

需求: “检测标志输出1且持续时间为1个cycle”

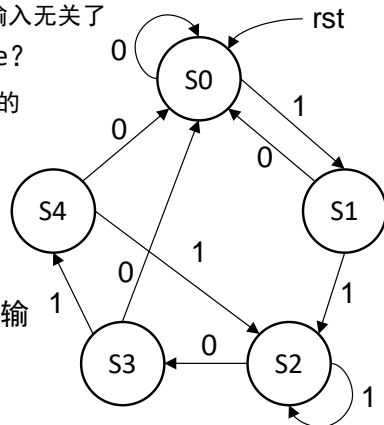
问题1: 什么情况下意味着检测到1101?

- 情况1: 在S3状态且data为1, 表明已经检测到了
- 情况2: 在S4状态。此状态下已经与输入无关了

问题2: 哪种情况可以满足1个cycle?

- 由于是同步时序电路, 因此任意状态的持续时间均是cycle的整数倍
- 本例中, S3或S4均只保持1个cycle
  - S2有可能会持续多个cycle

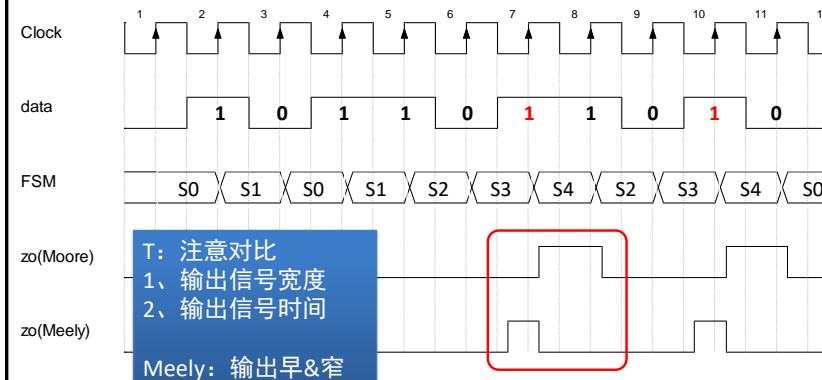
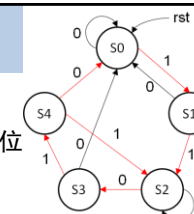
这一问题的实质是: 采用Moore型输出还是Meely型输出



## 对比两种输出信号的时序

时序分析

- Moore: 输出晚; 有效时间仅与状态相关, 以cycle为单位
- Meely: 可以早; 有效时间是输入与状态的AND



## STEP6: 产生输出信号

### 分析结论:

- 1、输出信号仅是状态寄存器的函数
- 2、进一步，输出信号仅当状态机为S4时输出1，否则输出0

Q2Q1Q0	zo
000	0
001	0
010	0
011	0
100	1
101	0
110	0
111	0

$$zo = Q2 \ \& \ !Q1 \ \& \ !Q0$$

Q  
如果在S3且data为1时zo输出为1，  
请给出zo的表达式

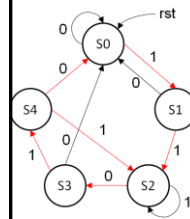
```

module monitor (clk,rst,data,zo);
    parameter S0=3'b000, S1=3'b001,
               S2=3'b010, S3=3'b011, S4=3'b100; // 状态编码
    input clk,rst,data;
    output zo;

    reg [2:0] state;

    assign zo=(state==S4)?1'b1:1'b0; // 输出信号
    always @(posedge clk or posedge rst)
        if (rst) state <= S0; // 复位时回到初始状态
        else
            case (state)
                S0: if (data==1'b1) state <= S1;
                     else state <= S0;
                S1: state <= (data==1'b1) ? S2 : S0;
                S2: if (data==1'b0) state <= S3;
                     else state <= S2;
                S3: state <= (data==1'b1) ? S4 : S0;
                S4: state <= (data==1'b1) ? S1 : S0;
                default: state <= S0;
            endcase
endmodule

```



T  
两种条件语句都可以



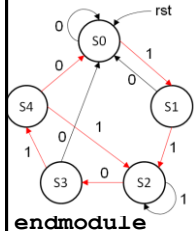
```

module monitor(clk,rst,data,zo);
    parameter S0=3'b000, S1=3'b001,
               S2=3'b010, S3=3'b011, S4=3'b100; // 状态编码
    input clk,rst,data;
    output zo;

    reg [2:0] state;

    assign zo=(state==S4); // 输出信号
    always @(posedge clk or posedge rst)
        if (rst) state <= S0; // 复位时回到初始状态
        else
            case (state)
                S0: state <= data ? S1 : S0;
                S1: state <= data ? S2 : S0;
                S2: state <= ~data ? S3 : S2;
                S3: state <= data ? S4 : S0;
                S4: state <= data ? S1 : S0;
                default: state <= S0;
            endcase
endmodule

```



T  
注意zo表达式和Sx表达式

```

module monitor(clk,rst,data,zo);
    parameter S0=3'b000, S1=3'b001,
               S2=3'b010, S3=3'b011,
    input clk,rst,data;
    output zo;

    reg [2:0] state;

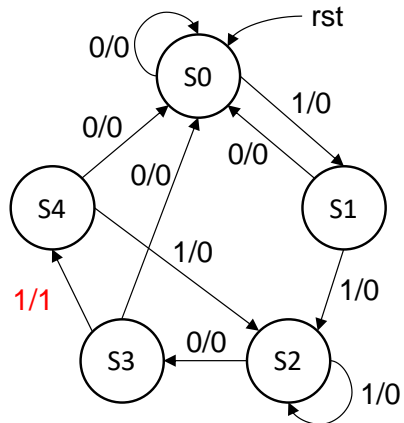
    assign zo=(state==S4);
    always @(posedge clk or posedge
        if (rst) state <= S0;
        else
            case (state)
                S0: state <= data ?
                S1: state <= data ?
                S2: state <= ~data ?
                S3: state <= data ? S4 : S0;
                S4: state <= data ? S1 : S0;
                default: state <= S0;
            endcase
endmodule

```

输入		现态			次态		
data	Q2	Q1	Q0	D2	D1	D0	
0	0	0	0	0	0	0	
1	0	0	0	0	0	1	
0	0	0	1	0	0	0	
1	0	0	1	0	1	0	
0	0	1	0	0	1	1	
1	0	1	0	0	1	0	
0	0	1	1	0	0	0	
1	0	1	1	1	0	0	
0	1	0	0	0	0	0	
1	1	0	0	0	0	0	
X	1	0	1	0	0	0	
X	1	1	0	0	0	0	
X	1	1	1	0	0	0	

T  
default: 可靠性设计, 确保状态机即使因某种特殊原因进入非法状态, 也必然最终能退出。

## 序列检测器的Mealy型有限状态机状态图



// 摩尔型

```
assign zo = (state==S4) ? 1'b1 : 1'b0 ;
```

// 米里型

```
assign zo = (state==S3) & (data==1'b0) ;
```

## 关于Moore与Meely的选择

□ 原则：输出信号选择Moore还是Meely，应取决于对输出信号的要求！

- ◆ 根据对输出信号的时间要求，来构造相应的输出信号表达式
- ◆ 不要纠结于Moore还是Meely的具体字眼

## [AD]Verilog建模异步复位和同步复位

### □ 区分要点：复位信号与时钟上升沿的关系

- ◆ 无关：异步复位。复位信号有效，则寄存器就被复位
- ◆ 相关：同步复位。复位信号有效且时钟上升沿时才能复位寄存器

### □ 假设：state为reg类型

```
always @(posedge clk or posedge rst)
```

```
    if (rst) state <= S0;
```

```
    else
```

```
        case (state)
```

```
            S0:
```

```
            S1:
```

```
            ...
```

```
always @(posedge clk)
```

```
    if (rst) state <= S0;
```

```
    else
```

```
        case (state)
```

```
            S0:
```

```
            S1:
```

```
            ...
```

#### 异步复位

复位信号通常由系统产生，一般仅在上电引起的全局复位时使用

#### 同步复位

复位信号既可以由系统产生，也可以由其他控制路产生。对于后者而言，通常是用于在某个特定状态清除寄存器值